Association for Information Systems
AIS Electronic Library (AISeL)

AMCIS 2012 Proceedings | Proceedings

Social Traps of Agile methods

Adarsh Kumar Kakar
Management Information Systems, University of Alabama, Tuscaloosa, AL, United States., askakar@crimson.ua.edu

Joanne Hale
Management Information Systems, University of Alabama, Tuscaloosa, AL, United States., jhale@cba.ua.edu

David Hale
Management Information Systems, University of Alabama, Tuscaloosa, AL, United States., dhale@cba.ua.edu

Social Traps of Agile methods

Adarsh Kumar Kakar
University of Alabama
askakar@crimson.ua.edu

Dr. Joanne Hale
University of Alabama
jhale@ua.cba.edu

Dr. David Hale
University of Alabama
dhale@ua.cba.edu

ABSTRACT

Social traps are situations within which individuals or groups face the prospect of adopting seemingly beneficial behaviors that have negative consequences over time or for a larger collective (Platt, 1973). While on the one hand, the Agile manifesto has been welcomed by many in the software developer community who often perceived formal processes as management generated inefficiency that gets in the way of productivity (Anderson, 2005). On the other hand proponents of formal plan-driven approaches argue that in spite of the lofty ideals and claims made, Agile development only postpone costs for the developing organization. Following the threads of this debate through a review of extant literature this conceptual study suggests that there is a real risk of developers using Agile methods falling into "social trap' of compromising the long-term goals for short-term gains.

Keywords

Social traps, Agile development

INTRODUCTION

The raging debate over plan-driven versus Agile methods of software development continues. The 2001 Agile manifesto (http://agilemanifesto.org), can be seen as a reaction to three decades of domination of planned processes for software development, since the waterfall model was first introduced in 1970s. The Agile manifesto has been welcomed by many in the software developer community who often perceived formal processes as management generated inefficiency that gets in the way of productivity (Anderson, 2005).

Proponents of the Agile methods argue that software development is a complex undertaking beset with many problems called “wicked problems” by Horst Rittel, an urban planner who pioneered the concept of issue-based information systems to facilitate the formulation and clarification of complex administrative decisions (Nerur and Balijepally, 2007). These problems tend to be unique and difficult to formulate, and solutions evolve continually as both the user and designer gain a greater appreciation of what must be solved. Agile Developers question the assumption that change and uncertainty can be controlled through a high degree of formalization and have discovered inadequacies in formal design that follow systematic procedures dictated by rigid processes (Nerur and Balijepally, 2007).

They pride themselves on highly productive, responsive, low ceremony, lightweight, tacit knowledge processes with little waste, adaptive planning and frequent iterative delivery of value (Anderson, 2005). By contrast, formal or plan-driven methods are often associated with conformance to plan, low trust environments, with command and control structures (Anderson, 2005). They require a big design up front approach with auditing of conformance and by implication punishment for non-conformance.

However, the proponents of plan-driven methods feel that Agile practices are rarely applicable in industrial settings and for industrial strength applications there is no alternative to formalized processes and plan-driven development. They argue that in spite of the lofty ideals and promises of Agile methodologies they are not delivering the goods. There is little scientific

support for the claims made by the Agile community (Dyba and Dingsoyr, 2008), and that whatever evidence presented in its favor is mainly anecdotal (Turk et al., 2002).

Further, the Agile approach follows the line, don't do anything today that you can postpone to tomorrow (Kajko-Mattsson, Lewis, Siracusa, Nelson, Chapin, Heydt, Nocks, and Snee, 2006). In other words the original development project is stripped to the bare minimum. This may be a good way of keeping tight deadlines, but it is not the best way to save costs. It is only postponing costs which then come later in the maintenance phase multiplied many times over. Software systems live long after they once go into production. The insignificant role that software architecture and documentation plays in Agile methodologies is an impediment to long-term software maintenance (Stammel, Durdik, Korgmann, Weiss and Koziolek, 2011).

It is argued that Agile approach builds software for short-term needs and does not provide the foundations for building enterprise level software. Integrative processes such as multi-project planning, overall software architecture and business process modeling are barely mentioned (Kajko-Mattsson, Lewis, Siracusa, Nelson, Chapin, Heydt, Nocks, and Snee, 2006). Agile methodologies meet local needs that may or may not benefit the organization as a whole. It is an instance of the classic sub-optimization problem in allocating organizational resources.

A review of extant literature suggests that there is more than a grain of truth to the arguments made by critics of Agile methods. This study draws upon the social trap theory to perform a socio-psychological analysis of the phenomenon to demonstrate how practitioners of Agile methods may be unwittingly falling into a "social trap" situation of sacrificing the long-term interests of an IS organization for near-term gains.

LITTERATURE REVIEW

Social trap theory has potential to shed light on real-world problems having roots in collective, selfish behavior-what Schelling (1971: 68) called "the frequent divergence between what people are individually motivated to do and what they might like to accomplish together."

What are social traps?

Platt (1973) identifies three major types of social traps:

1. One-person social trap or self trap: One person traps are situations which provide short-term benefits for the individual at long-term costs. For example cigarette smoking is a self trap because the short-run pleasure and social status associated with smoking, may lead to earlier death from smoking-induced cancer. More important, once this road has been taken it is very difficult to change; it may not be easy to quit smoking.

2. Missing hero trap or Volunteer dilemma: It describes situations whereby a group reaps benefits only if an individual or a requisite subgroup volunteers to incur personal costs (e.g., time, effort, money). Schelling's (1971) article provides an illustration. Consider the situation, on a summer Sunday evening, when thousands of cars are coming back from Cape Cod weekend on a two-lane road and a mattress falls unnoticed from the top of a station wagon and lies in the northbound lane. All of the cars behind, being uncertain, go around the mattress, waiting for the cars in the southbound lane to go by, and the result is a traffic jam that backs up for miles. Now who moves the mattress? The answer is generally, no one. People far back in the line do not know what the trouble is and cannot help. And the drivers close to the mattress are thinking only how to go around it quickly – and after they have spent so long in line they are damned if they will spend another several minutes, perhaps endangering themselves, to stop to move the thing. Those who have gone past, of course, of course no longer have the incentive for moving it. In such a situation it is true that sometimes a hero does come forward. However the lack of a heroic act lands the group in a persistent problem.

3. Collective trap: Collective traps are situations where the common pursuit of "individual goods leads to collective bads" (Platt, 1973). Garrett Hardin's (1968) article "The Tragedy of the Commons," cited the dilemma that arises when herdsmen graze cattle in a common pasture. Individual herdsmen seek to maximize their gain by grazing more and more cattle on the commons. Eventually, the pasture is overgrazed and the short-term individual gain-seeking behavior leads to long-term collective disaster.

Causes of Social traps

In general, social traps exist when "some of the costs or damages of what people do occur beyond their purview, and they either don't know or don't care about them" (Schelling, 1978). Failure to pursue collective interests in harvesting from a common resource pool is related to failure to pay sufficient cognitive attention to the effects of long-term consequences (Schroeder and Johnson, 1982). Koestler (1968) sees social traps as arising due to a conflict between the lower instinctive

brain and the recently evolved higher rational brain. One-person and collective social traps are time-delay traps resulting from the fact that the positive and negative reinforcements are separated; the rewards are immediate but the costs are delayed. Collective social traps may arise because individuals and groups or organizations are unable to cooperate owing to mutual distrust, even where cooperation would benefit all. People will cooperate only if they can trust that others will also cooperate (Rothstien, 2005). On the other hand, a missing hero trap occurs because unadulterated altruism is a rare bird.

Escaping Social traps

Like animal traps, social traps lead an unwary victim into the jaws of disaster with a tempting bit of bait, and, once the victim is caught, make escape extremely difficult (Constanza, 1987). However, many ways of avoiding social traps are suggested in literature (e.g., Cross & Guyer, 1980; Dawes, 1980; Edney, 1980; Messick & Brewer, 1983; Platt, 1973; Rutte, 1990; Van Lange, Liebrand, Messick, and Wilke, 1992). Some of the more salient ones are listed below:

Education: Individuals are more likely to sacrifice self-interest for cooperation when they have information about the underlying dynamics of a trap (Neidert and Linder, 1990; Rapoport, 1988). Education can be used to warn people about the detrimental long term impacts of short term actions such as through warning labels on cigarette packages.

Change the time-delay: By converting long term consequences (costs) into more immediate ones. Example includes solving the highway traffic jam problems through toll road corporations. Drivers were glad to pay the toll (immediate costs) for their immediate pleasure (rewards).

Add Counterreinforcers: Social incentives or punishments to encourage or discourage behaviors. Examples include punitive laws for traffic offenders and lower interest rates on loans for paying on time (high credit rating).

Set up Superordinate authority: A superordinate authority can be set up to present entrapments, allocate resources, mediate conflicts and redirect reinforcement patterns. For example governments can forbid or regulate certain actions that have been deemed socially inappropriate.

Changing a trap to a tradeoff: A trap may be changed to a tradeoff by imposing compensatory fees. For example, for resource stability, consumption above the optimum level may be taxed.

Get outside help for changing in reinforcement patterns: In many of our personal traps, a skilled outsider can help us see through our traps and change to a new pattern which we cannot easily on our own. Social security deductions is an example of an outside agency action to protect improvident man.

Communication amongst groups: Edney and Harper (1978) observed in a resource management study that team-member communication increased heroic acts. It permitted groups to anticipate traps and induce heroic acts.

Making individuals and groups to work together: The contact hypothesis (e.g., Allport, 1954), proposes that interaction across group boundaries attenuates intergroup conflict, permitting cooperation amongst groups. Social proximity brings greater opportunity for personal acquaintance and equal status, increasing the likelihood that differentiated interaction will displace group categorizations. The social identity theory (Tajfel & Turner, 1985) describes self-categorization processes that make boundaries between groups less salient cognitively for individual group members as a result of cooperative interaction.

Potential Social traps for projects using Agile methods

Agile methods focus on eliciting functional requirements. Its handling of non-functional requirements is ill defined (Paetsch, Eberlein and Maurer, 2003). Customers or users talking about what they want the system to do normally do not think about resources and long term issues such as maintainability, portability, or performance. Some non-functional requirements such as user interface or safety can be elicited during the development process and still be integrated. But most non-functional requirements should be known early in development because they can affect resource planning and quality. Not considering these issues early in the development lifecycle do have long term quality repercussions.

Agile methods and processes are not conducive to producing reusable artifacts and preclude developing generalized solutions even when it is clear that this could yield long-term benefits (Turk et al., 2002). The project/product-specific development in an Agile environment does not stress on the broader objectives of development of generalized solutions and other forms of reusable software. The processes used to build the reusable artifacts emphasize quality control because the impact of low quality is as wide as the number of applications that reuse the artifact. This may not be achievable in the Agile environment of informal reviews and paired programming. Agile developers view the introduction of formal evaluation and testing techniques for reusable artifacts as increasing project costs for something which may or may not be used in the future, yet their overall effect is to reduce the total cost of development. Studies have found a statistically significant relationship between reuse levels and lines-of-code productivity (Cusumano, MacCormack, Kemerev and Crandall, 2003).

The Agile view is that documents are not software and software development should develop software, not documents. This deliberate avoidance of documentation might also cause long-term problems for organizations adopting Agile methods. Documentation is used for sharing knowledge between people. A new team member will have many questions regarding the project. These can be answered by other team members or by reading and understanding "good" documentation. Asking other team members will slow down work because it takes some time to explain a complex project to someone. Talking is a very effective way to communicate but is also very ephemeral. Once the conversation is over, it is hard to quickly reach the precise information you are searching for. Documentation reduces knowledge loss when team members become unavailable as they move to another company or work on a new project. Lack of documentation frequently becomes an issue because most business softwares need to be maintained in the long run (Paetsch et al., 2003).

The small role that software architecture plays in Agile methodologies is another impediment to long-term software maintenance. Maintainability is a quality attribute that has to be built into the architecture of a system. The agile method seems to work well in a set of ideal conditions that do not hold true in many projects (Kajko-Mattsson et al, 2006) such as: the same people will be maintaining the software system throughout its entire life cycle, adding a feature is a simple task that should not have a major effect on the system and if it does have an effect, there is time to "sit back" and re-design or re-factor, either now or later.

This approach will only postpone costs which then come later in the maintenance phase. Quality has its price. It has to be paid sooner or later. The Agile methods approach follows the line, don't do anything today that you can postpone to tomorrow. In other words the original development project is stripped to the bare minimum. This may be a good way of keeping tight deadlines, but it is not the best way to save costs. The fact is that the costs of adding quality later are much higher than building in the quality from the start. Perhaps when users realize this, they will be less enthusiastic about the Agile practices (Kajko-Mattsson et al, 2006).

Agile processes emphasize the code focus. Proponents of Agile methods often argue that code is the only deliverable that matters, and marginalize the role of analysis and design models and documentation in software creation and evolution (Turk et al., 2002). The output of iterative development often resembles unmanageable "spaghetti code" that is difficult to maintain and integrate, a situation similar to the "code and fix" problems that the waterfall model was originally intended to correct. In "Long-term Life Cycle Impact of Agile Methodologies" (Kajko-Mattsson et al, 2006) one of the authors who participated in the debate lucidly explains the problem through a practical example: "In the projects that I have been involved in, which were conducted according to the Agile philosophy, such as the web portal for the German state of Saxony, the deadline was met and the budget was kept, but at what a price. The error rate was as high as 18 defects per kilo instructions. Through this approach they may build software that meet short-term, individual project needs, but that do not necessarily lead to software systems suitable for long-term "enterprise" software. The contractors purposely bid at a low cost and a tight deadline in order to win the project. However, they then spend the next two years debugging and cleaning up the product, something they should have done from the start. For that they are paid extra from the maintenance budget. My standpoint is that such Agile development is only postponing costs which then come later in the maintenance phase."

DISCUSSION

IS organizations, such the MIS department of an organization or a software development company, manage a basket of projects within budgeted resources. The projects include development projects, deployment projects and maintenance projects. New projects are continually added to the basket and existing projects prematurely discontinued or retired. Just as an individual project fulfills user requirements by building them into a IS product, an IS organization fulfills business goals through the products and services offered by the basket of projects. Also, just as different requirements have different user priorities, different projects have different business priorities. And just as the backlog of user requirements at the project level is actively managed by considering the smallest and most important of user requirements to be built into the product, the backlog of active projects at the IS organization level is actively and continuously managed to maximize the business value for the organization.

Projects thus compete for firm's resources. Although most literature on Agile project management have tended to be written with the overriding assumption that the projects are managed as single projects, it is a simplification and does not reflect the real situation. Projects are not islands by themselves. From the perspective of an IS organization they are managed as a portfolio of projects. These processes are explicit as in many large organizations or implicit as in many small organizations (Vähäniitty, Rautiainen. and Lassenius, 2010). The goal at the level of IS organization is not local

maximization at the project level, which is easier to achieve, but global maximization. It helps avoid situations in which a part of the system is optimized but the larger system is worse off as a result.

Agile projects are aligned to quickly meet the needs of its users. However, the agile principle of welcoming changing customer requirements has its own costs. While agreeing to every change suggested by the customers will keep them happy and contribute to project success it may impact the viability of other projects within the IS organization by expending more than their share of resources from the common pool. In addition, in the absence of a formal change control, this practice may proliferate redundancy by duplicating efforts made in other projects.

While Agile projects take the required resources from the common pool for their own consumption, they may not contribute significantly to other projects and to the IS organization as a whole. For example, a mature IS organization would be interested in maintaining a large growing repository of reusable components from its current and previous projects. However, in Agile projects, little support documentation coupled with time constraints and inadequate quality control mechanisms are an impediment to producing reusable artifacts which can be used by other projects to increase the overall productivity of the IS organization. The lack of adequate documentation is also responsible for inadequate transition support from development phase to deployment and maintenance phases. Unless deployment and maintenance is done by the same team that did the development, the absence of documentation will require support teams to invest additional resources for doing the same amount of work. The observation that support team members spend a major portion of their time reading source code and the comments they contain (De Souza, Anquetil and De Oliveira, 2005; De Souza, Anquetil and De Oliveira, 2007) are typical of the additional costs that are carried over from the development phase.

These situations represent typical cases of collective social trap where one group consumes resources from the common pool for its individual "good" but results in collective "bads" for the organization as a whole. Even if the deployment and maintenance were to done by the same team that did its development, it would still be a case of individual trap or self trap. The short term reward of developing the system quickly will have its long term costs in additional efforts required during deployment and maintenance phase due to information loss and "spaghetti" code. Under schedule pressure and lack of process discipline code refactoring suffers. Refactoring cannot be done effectively by one or two persons in Agile projects in the same way as design changes are managed by a few persons in plan-driven projects. In Agile projects each person has to refactor his own code due to different styles and lack of a uniform standard. To compound the problem it's very hard to tell if refactoring is actually happening. As a result the deployment groups and the maintenance types are like the incoming cars of the "mattress" example. They do not know the cause of the traffic jam until they come across the mattress. In the absence of disciplined processes the deployment and the maintenance groups can only expect "heroics" to save the day. They are the mercy of "missing heroes" of the development team to refactor the code and make them easier to read, understand and correct.

Taking advice from literature there are ways to avoid these social traps. Most Agile methods assume that development starts from scratch and ends with a release (Hanssen, Yamashita, Conradi, and Moonen, 2009). The Agile team should be educated about the requirements of life cycle phases beyond the development phase and the overall goals of the IS organization. Transition to subsequent phases should take in consideration the needs of deployment and maintenance teams for adequate documentation and refactoring of code in a disciplined manner. Projects should value-add even when they are prematurely discontinued or retired. Agile projects when discontinued do not leave behind worthwhile artifacts such as reusable code or lessons learnt. The only benefit is the tacit knowledge gained by the project team members. Even this learning may not get institutionalized for the benefit of others. It gets lost along with the team members when they leave the organization. However, education about these issues by itself may not be enough to escape the "social traps". Although it goes against the grain of the Agile philosophy, these educational programs may have to be coupled with counterreinforcers that will provide incentives or punishments to encourage or discourage behaviors. Good readable and adequately documented programs that meet the needs of all relevant stakeholders should be singled out as examples for others to emulate.

The other option is to bring Agile development under the preview of a superordinate authority (Platt, 1973). Regular quality assurance activities such as code inspections and process audits can be conducted to ensure that the projects do not sacrifice the long term consideration such as a well written, easily understood and reusable code for the short term considerations like quick delivery of working products. However this would be an expensive and bureaucratic way to overcome the social trap and would perhaps be a retrograde step. It would to some extent be a reversal to the plan-driven paradigm without reaping the benefits of Agile practices.

Yet another option is to promote cooperation amongst groups by involving the product managers and maintenance/ operations people who have a long term stake in the product, during the development process. Interaction across groups will enhance understanding and attenuate conflicts. It will narrow the divergence between in-group motivations and collective goals, thereby evading "social traps".

CONCLUSION

The Agile principles are insufficiently grounded in theory (Conboy and Fitzgerald, 2004). As a result the ongoing debates on the efficacy of Agile methods produce more sound than light. The "social trap" theory coupled with an analysis from the broader perspective of an IS organization, rather than the narrow domain of an individual project, offers interesting insights. The study reveals that although Agile methods provide rapid development of working products and responsiveness to users, lurking just below the surface are "social traps" that the IS organizations needs to be wary of. Agile projects are likely to fall victim to all three types of "social traps". While the project objectives may be achieved the effectiveness at the IS organization level are likely to be compromised. This study has implications for practice. In addition to identifying the potential "social traps" of using Agile methods, it provides guidance to IS organizations on how to escape them.

REFERENCES

1. Allport, G. W. (1954) The Nature of Prejudice, Reading, MA, Addison-Wesley.

2. Anderson, D. J. (2005) Stretching Agile to fit CMMI Level 3 - the story of creating MSF for CMMI ® Process Improvement at Microsoft Corporation, *presented at Agile2005 Conference*, http://agile2005.org/.

3. Conboy, K. and Fitzgerald, B. (2004) Toward a conceptual framework of agile methods: a study of agility in different disciplines, *in: Proceedings of XP/Agile Universe*, Springer Verlag.

4. Constanza, R. (1987) Social traps and environment policy, BioScience 37:407-412.

5. Cross. J. G. and Guyer, M. J. (1980) Social traps, Ann Arbor: University of Michigan Press.

6. Cusumano, M., MacCormack, A., Kemerev, C. F. and Crandall, B. (2003) Software Development Worldwide: the state of the practice, IEEE Software.

7. Dawes, R. M. (1980) Social dilemmas, Annual Review of Psychology, 31, 169-193.

8. De Souza, S. C., Anquetil, N., and De Oliveira, K. M. (2005) A study of the documentation essential to software maintenance, *Proceedings of the 23rd Annual International Conference on Design of Communication: Documenting and Designing for Pervasive Information (SIGDOC 2005),* Coventry, UK, 68-75.

9. De Souza, S. C., Anquetil, N., and De Oliveira, K. M. (2007) Which documentation for software maintenance?, *Journal of the Brazilian Computer Society*, 13(2), 31-44.

10. Dyba, T. and Dinsoyr, T. (2008) Empirical Studies of Agile Software Development: A Systematic Review, *Information & Software Technology*, 50(9-10), 833-859.

11. Edney. J. J. (1980) The commons problems: Alternative perspectives., *American Psychologist*, 35, 131-150.

12. Edney, J. J. and Harper, C. S. (1978) The commons dilemma: A review of contributions from psychology, *Environmental Management*, 2, 491-507.

13. Guntamukkala, V., Wen, H. J. and Tarn, J. M. (2006) An empirical study of selecting software development lifecycle models, *Human Systems Management*.

14. Hanssen, G.K, Yamashita A.F., Conradi, R. and Moonen, L. (2009) Maintenance and agile development: Challenges, opportunities and future directions. *In: Software Maintenance, IEEE International Conference on Software Maintenance*.

15. Hardin., G. (1968) The tragedy of commons. *Science*, 162, 1243.

16. Kajko-Mattsson, M., Lewis, G. A., Siracusa, D., Nelson, T., Chapin, N., Heydt, M., Nocks, J. and Snee, H (2006) Long-term Life Cycle Impact of Agile Methodologies, In *Proceedings of the 22nd IEEE International Conference on Software Maintenance (Philadelphia, Pennsylvania, USA, Sept., 2006). IEEE Computer Society*, 422-425.

17. Koestler, A.(1968) The ghost in the machine, New York: Macmilan.

18. Messick. D. M. and Brewer. M. B. (1983) Solving social dilemmas, In L. Wheeler and P. Shave (Eds.), *Review of personality and social psychology: 11-44, Beverly Hills: Sage*.

19. Neidert, G. P. M., and Linder, D. E. (1990) Avoiding social traps: Some conditions that maintain adherence to restricted consumption, *Social Behavior*, 5, 261-284.

20. Nerur, S. and Balijepally, V. (2007) Theoretical reflections on agile development methodologies, *Communications of the ACM*, 50 (3), 79–83.

21. Paetsch, F., Eberlein, A., and Maurer, F. (2003) Requirements Engineering and Agile Software Development, *Proceedings of the 12th IEEE international Workshops on Enabling*.

22. Platt, J. (1973. Social traps*, American Psychologist 2*, 641-651.

23. Rapoport, A. (1988) Provision of public goods and the MCS experimental paradigm, *American Political Science Review,* 79,148-55.

24. Rothstien, B. (2005) Social Traps and the Problem of Trust. Cambridge, UK: Cambridge University Press.

25. Rutte. C. G. (1990) Solving organizational social dilemmas, *Social Behavior*, 5, 285-294.

26. Schelling, T. (1971) On the ecology of micromotives, *The Public Interest*, 25, 59-98.

27. Schelling, T. C. (1978) Micromotives and macrobehavior, New York: Norton.

28. Stephens, M. and Rosenberg, D. (2003) Extreme Programming Refactored: The Case Against XP, Apress, Berkeley, CA. Technologies: Infrastructure for Collaborative Enterprises, 308 – 313.

29. Stammel, J., Durdik, Z., Korgmann, K., Weiss, R. and Koziolek, H. (2011) Software Evolution for Industrial Automation Systems: Literature Overview, *Karlsruher Institut für Technologie*.

30. Schroeder, D. A. and Johnson, D. E. (1982) Utilization of information in a social-trap situation, *Psychological Reports*, 50, 107–113.

31. Tajfel, H. and Turner, J. C. (1985) The social identity theory of intergroup behavior, In S. Worchel & W. G. Austin (Eds.), Psychology of intergroup relations (2nd ed., 7-24). Chicago: Nelson-Hall.

32. Turk, D., France, R. and Rumpe, B. (2002) Limitations of Agile Software Processes, *in Proceedings of Third International Conference on eXtreme Programming and Agile Processes in Software Engineering*, Italy.

33. Vähäniitty, J. Rautiainen, K. and Lassenius, C. (2010) Small software organisations need explicit project portfolio management, *IBM J. RES. & DEV.,* 54, 1-12.

33. Van Lange, P. A. M.. Liebrand. W. B. G., Messick, D. M. and Wilke, H. A. M. (1992) Introduction and literature review, In W. Liebrand, D. Messick, & H. Wilke (Eds.), Social dilemmas: Theoretical issues and research findings (pp. 3-28), Oxford, England: Pergamon Press.